[0018] Embodiments of the present invention provide a solution to the interactive analysis of attack graphs. First, the embodiments may represent in a relational model the necessary inputs including network configuration and domain knowledge. The embodiments may then generate attack graph(s) using relational queries, which can either be materialized as relations or simply left as the definition of relational views. The latter case is especially suitable for large networks where materializing the complete attack graph may be prohibitive. Second, analyses of attack graphs may be realized as relational queries. The interactive analysis of attack graphs should now be possible, because administrators can immediately pose new queries based on the outcome of previous analyses. Finally, as a side-benefit, the performance of an analysis can usually be transparently improved by the mature optimization techniques available in most relational databases.

[0019] Attack graphs represent the knowledge about the inter-dependency between vulnerabilities. Model checking was first used to decide whether a goal state is reachable from the initial state and later used to enumerate all possible sequences of attacks connecting the two states. However, the number of attack sequences is potentially exponential, leading to high complexity. A more compact representation based on the monotonicity assumption (that is, an attacker never relinquishes an obtained capability) may be used. The new representation may keep exactly one vertex for each exploit or condition, leading to attack graphs of polynomial size.

[0020] Analyses of attack graphs have been used for different purposes in defending against network intrusions. Minimal critical attack set analysis finds a minimal subset of attacks whose removal prevents attackers from reaching a goal state. However, the attacks in a minimal critical attack set are not necessarily independent, and a consequence may not be removed without removing its causes. This observation leads to the minimum-cost hardening solution, which is a minimal set of independent security conditions. Finding the minimum set of attacks leading to given goals may be computationally infeasible, whereas a minimal set may be found in polynomial time. All attacks involved in at least one of such minimal sets of attacks may also be enumerated. Finally, in exploit-centric alert correlation, attack graphs may assist the correlation of isolated intrusion alerts.

[0021] The afore-mentioned analysis of attack graphs is largely based on proprietary algorithms. However, as mentioned earlier, this may delay a new analysis and make interactive analysis impossible. The disclosed embodiments remove this limitation and enable interactive analysis of attack graphs. On the other hand, decision support systems, such as on-line analytical processing (OLAP) [7], have been used for interactive analysis of data for a long time. However, an analyst there is usually interested in generalized data and statistical patterns, which is different from the analysis of attack graphs.

[0022] Attack graphs are usually visualized as a directed graph having two type of vertices, exploits and security conditions (or simply conditions). An exploit is a triple (h_s, h_d, v), where h_s and h_d are two connected hosts and v is a vulnerability on the destination host h_d. A security condition is a pair (h, c) indicating the host h satisfies a condition c relevant to security (both exploits and conditions may involve more hosts, for which the model can be easily extended).

[0023] An attack graph preferably has two types of edges denoting the inter-dependency between exploits and conditions. First, a require relation is a directed edge pointing from a condition to an exploit. The edge means the exploit cannot be executed unless the condition is satisfied. Second, a imply relation points from an exploit to a condition. This means executing the exploit should satisfy the condition. Notice that there is usually no edge between exploits (or conditions). Example 1 illustrates the concept of attack graph.

Example 1

[0024] FIG. **1**A depicts a running example of an attack graph with the exploits shown as ovals. FIG. **1**B illustrates an example of a simplified version the attack graph with the exploits shown as triplets. In FIG. **1**B, x denotes the existence of a vulnerability SADMIND BUFFER OVERFLOW (Nessus ID 11841), y the user privilege, and A the exploitation of that vulnerability. The attack graph shows an attacker having user privilege on host **3** may exploit the vulnerability on hosts **1** and **2** and obtain user privilege on the hosts.

[0025] Two important aspects of attack graphs are as follows. First, the require relation should always be conjunctive whereas the imply relation should always be disjunctive. More specifically, an exploit should not be realized until all of its required conditions have been satisfied, whereas a condition may be satisfied by any one of the realized exploits. Second, the conditions may be further classified as initial conditions (the conditions not implied by any exploit) and intermediate conditions. An initial condition may be independently disabled to harden a network, whereas an intermediate condition usually cannot be [12].

[0026] A Relational Model for Attack Graphs. In the relational model, the complete attack graph may be left as the result of a relational query (i.e. not explicitly represented in our model). The result to the query may be materialized, or the query can simply be left as a view. Such flexibility may be important to large networks where materializing the complete attack graph may be prohibitive. Two inputs may be modeled, the network configuration (vulnerabilities and connectivity of the network) and the domain knowledge (the interdependency between exploits and conditions), as illustrated in Example 2. The domain knowledge may be available in tools like the Topological Vulnerability Analysis (TVA) system developed at George Mason University, which covers more than 37,000 vulnerabilities taken from 24 information sources including X-Force, Bugtraq, CVE, CERT, Nessus, and Snort [8]. On the other hand, the configuration information including vulnerabilities and connectivity may be easily obtained with tools such as the Nessus scanner [5].

Example 2

[0027] FIG. **2** shows an example of a network configuration and domain knowledge used in generating the attack graph in Example 1. The left-hand side of FIG. **2** shows the connectivity between three hosts, and initially hosts **1** and **2** satisfy the condition x and host **3** satisfies y. The right-hand